CSC338. Homework 3

Due Date: Wednesday January 29, 9pm

Please see the guidelines at https://www.cs.toronto.edu/~lczhang/338/homework.html

What to Hand In

Please hand in 2 files:

- Python File containing all your code, named `hw3.py`.
- PDF file named `hw3_written.pdf` containing your solutions to the written parts of the assignment. Your solution can be hand-written, but must be legible. Graders may deduct marks for illegible or poorly presented solutions.

If you are using Jupyter Notebook to complete the work, your notebook can be exported as a .py file (File -> Download As -> Python). Your code will be auto-graded using Python 3.6, so please make sure that your code runs. There will be a 20% penalty if you need a remark due to small issues that renders your code untestable.

Make sure to remove or comment out all matplotlib or other expensive code before submitting your homework!

Submit the assignment on **MarkUs** by 9pm on the due date. See the syllabus for the course policy regarding late assignments. All assignments must be done individually.

```
import math
import numpy as np
```

Question 1

Part (a) – 4 pt

Solve the following system $A\mathbf{x} = \mathbf{b}$ using Gauss Elimination **by hand**. Show all your steps. Show all your steps in your pdf writeup.

$$A = \begin{bmatrix} 1 & 0 & -2 \\ -1 & 1 & 1 \\ 0 & 2 & -1 \end{bmatrix}, b = \begin{bmatrix} -1 \\ 2 \\ 3 \end{bmatrix},$$

Part (b) – 2 pt

Using your work from part (a), find the LU factorization of A. Show all your steps in your pdf writeup.

Question 2

For this question, you can refer to and use any of the code that we wrote together in tutorial 3:

```
# Code from tutorial 3

def backward_substitution(A, b):
    """Return a vector x with np.matmul(A, x) == b, where
        * A is an nxn numpy matrix that is upper-triangular and non-singular
        * b is an nx1 numpy vector
    """
    n = A.shape[0]
    x = np.zeros_like(b, dtype=np.float)
```

```
    for i in range(n-1, -1, -1):
        s = 0
        for j in range(n-1, i, -1):
            s += A[i,j] * x[j]
        x[i] = (b[i] - s) / A[i,i]
    return x

def eliminate(A, b, k):
    """Eliminate the k-th row of A, in the system np.matmul(A, x) == b,
    so that A[i, k] = 0 for i < k. The elimination is done in place."""
    n = A.shape[0]
    for i in range(k + 1, n):
        m = A[i, k] / A[k, k]
        for j in range(k, n):
            A[i, j] = A[i, j] - m * A[k, j]
        b[i] = b[i] - m * b[k]

def gauss_elimination(A, b):
    """Return a vector x with np.matmul(A, x) == b using
    the Gauss Elimination algorithm, without partial pivoting."""
    for k in range(A.shape[0] - 1):
        eliminate(A, b, k)
    x = backward_substitution(A, b)
    return x
```

Part (a) – 3 pt

Write a function `forward_substitution` that solves the lower-triangular system $Ax = b$.

Hint: This function should be very similar to the `backward_substitution` function from the tutorial.

```
def forward_substitution(A, b):
    """Return a vector x with np.matmul(A, x) == b, where
        * A is an nxn numpy matrix that is lower-triangular and non-singular
        * b is a nx1 numpy vector

    >>> A = np.array([[2., 0.],
                      [1., -2.]])
    >>> b = np.array([1., 2.])
    >>> forward_substitution(A, b)
    array([ 0.5 , -0.75])
    """
```

Part (b) – 4 pt

Write a function `elementary_elimination_matrix` that returns the k-th elementary elimination matrix (M_k in your notes).

You may assume that `A[i,j] = 0 for i > j, j < k - 1`. (The subtraction in $k - 1$ is because in Python, indices begin at 0.)

```
def elementary_elimination_matrix(A, k):
    """Return the elements below the k-th diagonal of the
    k-th elementary elimination matrix.

    (Do not use partial pivoting, since we haven't
    introduced the idea yet.)
```

```
    Precondition: A is an nxn numpy matrix, non-singular
                  A[i,j] = 0 for i > j, j < k - 1

    As always, these examples are for your understanding only.
    The actual Python output might differ slightly.

    >>> A = np.array([[2., 0., 1.],
                      [1., 1., 0.],
                      [2., 1., 2.]])
    >>> elementary_elimination_matrix(A, 1)
    np.array([[1., 0., 0.],
              [-0.5, 1., 0.],
              [-1., 0., 1.]])
    >>> A = np.array([[2., 0., 1.],
                      [0., 1., 0.],
                      [0., 1., 2.]])
    >>> elementary_elimination_matrix(A, 2)
    np.array([[1., 0., 0.],
              [0., 1., 0.],
              [0., -1., 1.]])
    """
```

Part (c) – 4pt

Write a function `lu_factorize` that factors a matrix A into its upper and lower triangular components. Use the function `elementary_elimination_matrix` as a helper.

```
def lu_factorize(A):
    """Return two matrices L and U, where
            * L is lower triangular
            * U is upper triangular
            * and np.matmul(L, U) == A
    >>> A = np.array([[2., 0., 1.],
                      [1., 1., 0.],
                      [2., 1., 2.]])
    >>> L, U = lu_factorize(A)
    >>> L
    array([[1. , 0. , 0. ],
           [0.5, 1. , 0. ],
           [1. , 1. , 1. ]])
    >>> U
    array([[ 2. ,  0. ,  1. ],
           [ 0. ,  1. , -0.5],
           [ 0. ,  0. ,  1.5]])
    """
```

Part (d) – 2pt

Write a function `solve_lu` that solves a linear system Ax=b by * factoring $A = LU$ (using the `lu_factorize` function) * solving $Ly = b$ using forward substitution (using the `forward_substitution` function) * solving $Ux = y$ using backward substitution (using the `backward_substitution` function)

```
def solve_lu(A, b):
    """Return a vector x with np.matmul(A, x) == b using
    LU factorization. (Do not use partial pivoting, since we haven't
    introduced the idea yet.)
    """
```

Part (e) – 5 pt

Write a function `invert_matrix` that takes an $n \times n$ matrix `A`, and computes its inverse by solving n systems of linear equations of the form $Ax = b$.

You can use the functions that you wrote earlier, but note that you **will** be graded on efficiency. In particular, avoid writing code that repeats a computation unnecessarily.

```
def invert_matrix(A):
    """Return the inverse of the nxn matrix A by
    solving n systems of linear equations of the form
    Ax = b.

    >>> A = np.array([[0.5, 0.],
                      [-1., 2.]])
    >>> invert_matrix(A)
    array([[ 2. , -0. ],
           [ 1. ,  0.5]])
    """
```

Question 3

Part (a) – 4 pt

Prove that the matrix

$$A = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

has no LU facotirzation. That is, there are no lower triangular matrix L and upper triangular matrix U such that $A = LU$.

Include your proof in your pdf writeup.

Part (b) – 2 pt

Show that for an elementary matrix $M_k = I - \mathbf{m}\mathbf{e}_\mathbf{k}^\mathbf{T}$, we have $M_k^{-1} = I + \mathbf{m}\mathbf{e}_\mathbf{k}^\mathbf{T}$. (Property 4 of the elementary elimination matrix from your lecture notes)

Include your solution in your pdf writeup.